Towards Interactive Programming of Modular Robots

Bordignon, Mirko; Støy, Kasper; Christensen, David Johan; Schultz, Ulrik Pagh

Published in:
IROS Workshop on Self-Reconfigurable Robots & Systems and Applications

Publication date:
2008

Document Version
Publisher's PDF, also known as Version of record

Towards Interactive Programming of Modular Robots

M. Bordignon, K. Stoy, D.J. Christensen and U.P. Schultz

Abstract— **Programming modular robots is typically a difficult and time-consuming task. In this position paper we describe our vision for a programming platform supporting interactive and incremental development of autonomous controllers for the ATRON and Odin modular robots. Currently, the platform consists of execution environments for the respective robots, a simulator, and a user-interface that allows for straightforward remote control using simple gait control tables.**

We envision interactive programming of distributed controllers being done by incrementally transitioning from simple remote control of a specific robot configuration to autonomous control of a whole class of similar robot configurations. This incremental transition relies on three key elements: (1) using symbolic names to globally identify what components (modules, specific sensors, actuators, etc.) to control, (2) describing behaviors using a distributed scripting language, and (3) dynamically updating code in the running system. We conclude that in this way we may possibly ease the task of programming modular robots by combining the highly interactive development style of the on-line approach with the expressiveness and ease of use of a scripting language.

I. Introduction

Programming of modular robots is mainly for programmers with masochistic tendencies: a modular robot is from a programming point view a distributed embedded system with dynamically evolving network topology where each node typically has limited computational power. In addition a modular robot is obviously also a robot which means that it has to control its actuators to act as a coherent whole based on rich sensor information from the environment. Unlike the typical case for modern programming languages, existing approaches for programming modular robots often fail to provide an immediate feedback that can help the programmer interactively develop the controller. We therefore think programming of modular robots requires a radical new approach.

In this paper we propose the idea of incrementally evolving simple remote-control behaviors into a fully autonomous controller for a modular robot. To support our approach we developed an initial prototype of a simple interactive tool, named Modular Commander, that allows the user to interactively experiment with modular robots using remote control. Modular Commander currently supports programming the Odin modular robot and the ATRON self-reconfigurable robot through a user-interface that can send commands wirelessly to any module in the system that then interprets and executes the command. Modular Commander allows us to implement gaits and other simple open-loop controllers that can be executed centrally from the interface. To allow autonomous controllers to be developed incrementally starting from this remote control scenario, we plan to extend Modular Commander into a complete interactive programming platform comprising a GUI that provides control of and sensor feedback from the robot, a high-level distributed programming language and corresponding virtual machine allowing behaviors to be incrementally added to the robot, and a symbolic naming scheme that allows the programmer to abstract over the concrete physical shape of the robot. This extension is work in progress, and our vision is to develop a complete system for interactive on-line programming and debugging of modular robots.

The authors are with The Modular Robotics Lab, Maersk Mc-Kinney Moller Institute, Faculty of Engineering, University of Southern Denmark, Denmark. [mirko,kaspers,david,ups]@mmmi.sdu.dk

This work was supported by The Danish Council for Technology and Production.

II. Experimental Platform

Our experimental tools are primarily designed to fit the ATRON and Odin robots, though in principle they can easily be adapted to other types of modular robots. The execution environment for the embedded controllers makes minimal assumptions about the modules' CPU, therefore representing a viable design for most hardware platforms. Our initial prototype of a PC-side command tool, named Modular Commander, uses gait tables to remote control the modules, allowing the user to interactively experiment with behaviors. Last, our USSR simulator [1], [2] already supports several types of modular robots and can easily be further extended.

A. Hardware

The ATRON self-reconfigurable modular robot is a 3D lattice-type system [3]. Figure 1 shows an example ATRON car robot built from 7 modules. An ATRON module has one degree of freedom, is spherical and is composed of two hemispheres which can be rotated relatively to each other. A module may connect to neighbor modules using its four actuated male and four passive female connectors. The connectors are positioned at 90 degree intervals on each hemisphere. Eight infrared ports, one below each connector, are used by the modules to communicate with neighboring modules and sense distance to nearby objects. The hardware is controlled by two Atmel ATMega128 micro-controllers, one on each hemisphere, communicating between them through an RS-485 serial link. Each CPU has 128Kbytes of flash memory for storing programs and 4Kbytes of RAM for use during program execution.

The Odin modular robot is a heterogeneous system composed of link and joint modules [4]. A link connects to a joint at each end of its cylindrical body, and the arrangement of the connections on the joints determines the lattice

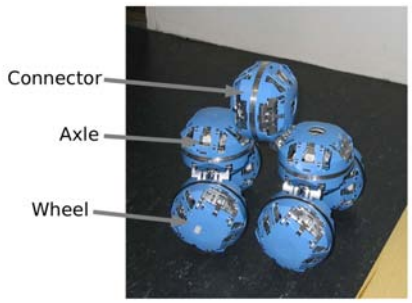


Fig. 1. The ATRON self-reconfigurable robot. Seven modules are connected in a car-like structure.

structure of the robot. In the current revision of the system the joints are passive elements of the mechanical structure, allowing both power sharing and communication over RS-485 among different link modules. Different functionalities, such as like passive structural support, sensing, actuation and power storage, are implemented by different kinds of link modules [5]. The electronics of the link modules consists of a general board common to all the different links and a specific one related to the different incorporated functionalities. The control software runs on the CPU of the general board, an Atmel AT91SAM7S micro-controller with 256Kbytes of program memory and 16 Kbytes of RAM.

B. Low-level execution environment

A common design underlies the structure and implementation of the embedded software running on the micro-controllers of both ATRON and Odin systems. According to our development experience [6], a conventional operating system providing a whole range of features like hardware abstraction, resource sharing and concurrency management can be overly constraining. Therefore we aimed at providing just a lightweight execution environment with part of the features a traditional operating system offers, with the rest ideally addressed by reusable application-level components.

As in most modular robotic systems, the ATRON processing units are cheap and therefore resource-constrained in order to keep the system reasonably simple and potentially cost-effective. What these CPUs are responsible of is not as simple though, given the multiple communication channels to manage and actuators to control. In our experience this can easily lead to peaks of heavy interrupt load which can render the system temporarily unresponsive.

We had two main objectives driving the design and implementation of an execution environment to support programs running directly on the modules. One was to manage the hardware so as to better deal with interrupt intensive situations and to provide useful high-level abstractions. The other was not to enforce a particular programming paradigm, allowing the adoption of the most suitable one for the particular application at hand. We chose to adopt a minimal event-driven kernel: interrupt handlers are limited to the strictly necessary operations, while the rest of the computation is performed inside run-to-completion functions scheduled to execute later, a mechanism similar to bottom-half handling [7]. To this end, we wrote a port of TinyOS [8] for the ATRON hardware which provides this functionality [6]. Another benefit of this implementation, which supports our second design goal, is the conceptual split of long running operations in two parts: a system call requesting the start of an operation (e.g. a connector extension), and a user defined callback executed upon completion (e.g. the connector is fully extended). As a side aspect, the low memory requirements (a single stack is allocated in RAM) and low execution overhead (no context switches) match well the tendency towards simpler, cheaper robot components highlighted by recent proposals [9]. This model allows a natural implementation of reactive style controllers, whose logic can be implemented as a simple list of callbacks. More complex controllers can make use of application-level constructs built over this simple kernel: for example, we show how a lightweight threading abstraction can be selectively employed to achieve hybrid control strategies [6].

By providing a system interface to the robot which explicitly exposes request and completion phases we also allow for transparent migration of selected components to a hardware implementation, a design possibility our group is actively exploring [10]. The semantics is in fact the same in both cases, with request calls equivalent to hardware request operations (e.g. setting values on input pins or a writing a register), and completion callbacks akin to interrupt requests.

The embedded software environment for the Odin modular robot is being developed following the same principles, with a similar event-driven system at its core derived in this case from Contiki [11] due to CPU port availability issues. The interface to the ATRON and Odin hardware consists overall of a set of plain ANSI C functions for both the system calls and the completion callbacks.

We are actively using this C interface in our experimental sessions. It proved to be effective for the development of autonomous controllers in stand-alone, long-runnning experiments [12]; it provides the foundations over which we are developing a scripting language interpreter, as described in Section IV-B.

C. Simulation

Simulation is a useful tool for experimenting with e.g. complex behaviors or large numbers of modules before the time-consuming setup of physical experiments. Simulation of the ATRON and Odin robots is supported by our simulator for self-reconfigurable robots. This simulator, named the "Unified Simulator for Self-Reconfigurable Robots" (USSR), is designed to support a wide variety of self-reconfigurable robots [1], [2], which currently includes the ATRON, Odin, and M-TRAN systems [3], [13], [14]. It is based on a physics engine and hence allows simulation of dynamic interaction with the environment, such as friction and object manipulation, but is also precise enough to simulate self-reconfiguration. The simulator is implemented in Java but provides a lightweight interface for controllers written in ANSI C.

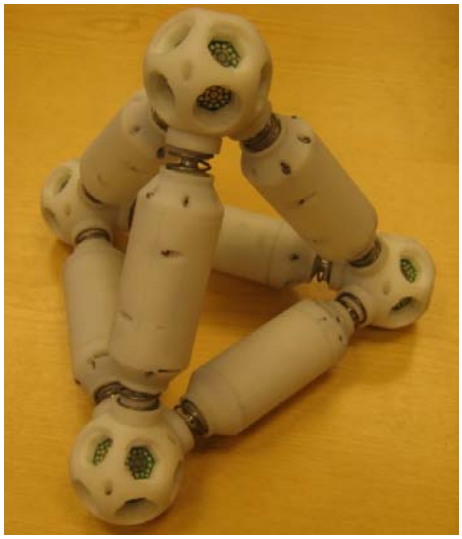


Fig. 2. The Odin reconfigurable robot in a tetrahedron configuration.

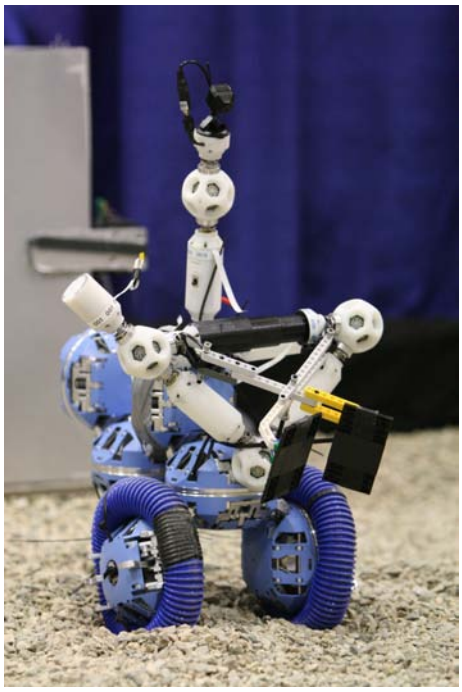


Fig. 3. ATRON-Odin hybrid robot

III. Modular Commander

We now describe Modular Commander, our initial prototype of a tool for remote control of modular robots, and assess its advantages and disadvantages compared to a standard off-line approach to programming modular robots.

A. Motivation

Development of controllers for ATRON and Odin robots has typically been done using a classical off-line approach where a program is implemented on a workstation and flashed to each of the modules of the robot using a JTAG connector, after which the robot is activated and the behavior can be investigated. If the behavior is incorrect the programmer must then go back to the program, modify it appropriately, and repeat the same process. An alternative solution that allows the programmer to start to work immediately and interactively with the behavior is using a tool to program the robot using gait tables [15]. Until now, the lack of such a tool for the ATRON and Odin robots has complicated experimenting with the physical capabilities of these modules.

The concrete motivation for developing Modular Commander was the ICRA'08 Planetary Contingency Challenge. The challenge basically concerned quickly developing and deploying a robotic solution to various tasks, using only a limited number of resources (parts that can fit in a suitcase). Tele-operation was explicitly permitted, which put the primary focus on developing specific robot configurations and providing a means of efficiently controlling this robot. One of the robots developed by the USD Modular Robotics Lab team was a mobile robot with a gripper and a video camera, shown in Figure 3. This robot was built by combining ATRON and Odin modules using tape etc. so that each type of robot can be used for the task to which its morphology is the best suited: ATRON provides wheels and rotating actuators whereas Odin provides bending structures and manipulators. The ATRON and Odin robots are independent systems that do not communicate with the other but rather are remote-controlled over a wireless link from the same computer using Modular Commander. During programming each module is assigned a unique ID stored in non-volatile (e.g. EEPROM) memory.

B. Implementation

The Modular Commander GUI allows the programmer to connect to an ATRON and/or Odin robot over a serial connection (currently only a single robot of each type).[1] A command-line interface and GUI allow predefined commands to be sent to each robot, where they are in turn executed by an interpreter program. All communication between the modules of the robot is done using a broadcast protocol, with a single module having a wired or wireless serial connection to the PC serving as gateway. Similarly, debugging output from the robots is displayed in a window. The commands allow the programmer to (1) define a mapping between virtual IDs used for programming and physical IDs stored in modules (allowing modules to be transparently replaced), (2) send a single command to each of the modules of the robot (referred to as a pose), and (3) send a sequence of poses defined in a gait table. In effect, the hybrid ATRON/Odin robot can be remote-controlled through a single interface. Defining ID mappings, poses, or gaits is simply done in a textual file, either by directly writing the corresponding command code as a number or by using a symbol (defined in a separate file) that resolves to this number.

[1] We are currently working on supporting Modular Commander in USSR, which will allow the developer to transparently switch between physical and virtual robots during experimentation.

C. Assessment

Although very simplistic and obviously an early prototype, Modular Commander was in practice highly useful as a tool for controlling a hybrid modular robot. The simplicity of assigning global IDs to the modules combined with remote control allows the programmer to very easily experiment with new hardware configurations. Reliance on global IDs for direct control is however not desirable in the long run: replacing modules or later rebuilding the robot becomes complicated, and the behavior of the robot is dependent on the orientation of the connectors of each module, which is tedious and error-prone to maintain when rebuilding the robot. Remote control is obviously inappropriate if we consider autonomous behavior as a goal; autonomous behavior is a requirement in many scenarios, and basically requires the robot to react to conditions pertaining to the environment and the internal state of the robot. To improve the situation, we would need to enable the programmer to incrementally replace global IDs with a local means of selecting what behaviors to activate on a give module, combined with a means of autonomously triggering behaviors that run throughout the modules of the robot.

IV. Towards Interactive Programming

Our long-term goal is to extend our platform into an interactive development environment for programming autonomous modular robots. While we have not yet achieved this goal we believe that we already have some elements of a solution that may enable us to reach this ambitious goal. These elements are the focus of this section.

Our basic idea is that the programmer develops a set of rules that identify and label the hardware components (modules, sensors, actuators, etc.) that are used in the controller. These labelled components are then used in a simple scripting language that to some extent hides the distributed nature of the robot and, through the labeling rules, changes in the physical configuration. We plan to extend the scripting language with the concepts of time and space, but as a first try these elements are hidden from the programmer. The resulting controller needs to be distributed to the modules, which is currently done by individually downloading it to the module CPUs. As an alternative, we developed a proof-of-concept virtual machine that allow programs to be distributed more easily as byte-code [16], and we are currently in the process of fully implementing this virtual machine on top of the execution environment described in section II-B. We now examine each of the individual pieces in turn.

A. Identifying and labelling components

We identify and label components using a set of rules that place constraints on the immediate context of a module in terms of the number of connections and the labels of neighboring modules [17]. For example, in the configuration depicted in Figure 1, if a module has only one neighbor it is labelled *wheel*. As an extension, a network-level service can be used to keep track of the relative compass direction and 3D-position of each module, which allows the programmer to e.g. differentiate between the left and right wheels of a car robot [16], [18], and provides a means of identifying e.g. the forward proximity sensor.

B. Adding behaviors to components

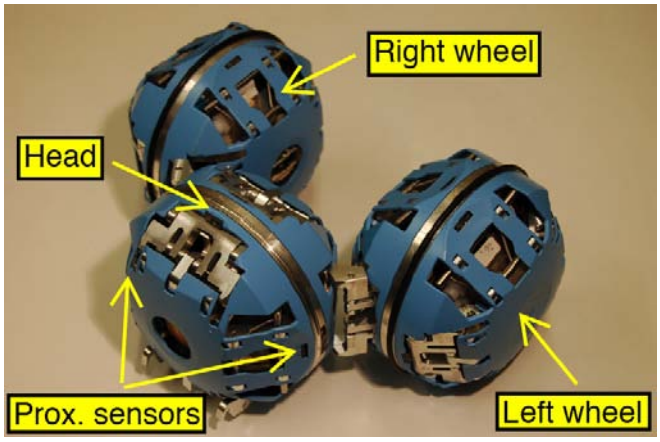
The labelled components can be used in many different ways: it is possible for instance to use them to implement role-based control [19]. Here we will use them to program a small configuration of modules whose behaviors are tightly coupled.

We refer to the set of modules that we program in our simple scripting language as a *behavior group*. Each statement in this language is prefixed which the label of the components that are going to perform the statement. The same script-based program is interpreted on all modules in the behavior group, but only statements involving components of the module on which the code is running are actually executed. The redundant execution of code makes it possible to recover from error states: it may for instance be possible to instruct modules that have been reset where to restart from, though currently we are not exploiting this robustness. The other feature of this approach is that application-specific code is localized in one place, making it easier to get correct sequencing and dependencies of a group behavior spanning several modules. Currently synchronization is implicitly enforced if during the execution of a group-behavior a condition is encountered which is based on a component not located on the module. In this case the module will pause the execution, and when the module owning that specific component reaches this statement it will broadcast the needed information throughout the behavior group. While this synchronization mechanism has been sufficient so far, explicit synchronization statements are probably needed for more complex tasks.

In order to test this approach we have implemented a group-behavior that allows two ATRON car robots consisting of three modules to dock with each other [20]. Figure 4 shows a small program fragment from this experiment, to give a feeling for the language. This program allows a car, also shown in the figure, to turn on the spot if one of its two frontal proximity sensors are activated. In this example the components have been labelled by hand as indicated in the figure, but we plan to have them labelled automatically as described in Section above. In this program the wheels are initially told to rotate to move the car forward. In the following if-statement all modules wait for the distribution of the proximity sensor states and depending on the result either make the left wheel reverse for two seconds or iterate one more time.

C. Code distribution

Currently each module is manually reprogrammed using a JTAG connector, which severely hamper interactive experimentation. It is therefore essential to provide a means for dynamically distributing new behaviors to the modules of the robot. Such code distribution concerns both distributed and local behaviors, should be done dynamically while the robot




```
LEFT_WHEEL, CMD_ROTATE_CLOCKWISE,
RIGHT_WHEEL, CMD_ROTATE_COUNTER_CLOCKWISE,

CAR, CMD_REPEAT, FOREVER,
  CAR, CMD_IF_OBJ_PRESENT,
       PROX_FRONT_LEFT | PROX_FRONT_RIGHT,
  CAR, CMD_IF_OBJ_PRESENT_BEGIN,
    LEFT_WHEEL, CMD_ROTATE_COUNTER_CLOCKWISE,
    CAR, CMD_DELAY, 20, 0,
    LEFT_WHEEL, CMD_ROTATE_CLOCKWISE,
  CAR, CMD_IF_OBJ_PRESENT_END,
  CAR, CMD_DELAY, 2, 0,
CAR, CMD_REPEAT_END
```

Fig. 4. A small piece of script code that allows the three-module ATRON car shown at the top to reverse if its front sensors are activated. The components the labels refer to are shown in the figure above except for *car*, which refers to all modules in the behavior-group.

is running, and should minimize the amount of communication required. To this end, we can use a virtual machine that enables small bytecode programs to move throughout a structure of ATRON modules [16]. The prototype virtual machine, named DCD-VM, has an instruction set that is dedicated to the ATRON hardware and includes operations that are typically required in ATRON controllers. The virtual machine supports a concept we refer to as *distributed control diffusion:* controller code is dynamically deployed to those modules where a specific behavior is needed.

D. Tying the pieces together

At the time of writing the elements just described are not integrated, which would obviously be the first step to take. The next step is to integrate these elements with the Modular Commander user-interface. We imagine that the design of rules for labelling can be done interactively with the modules, physical or simulated, giving feedback to the user about which components are selected based on the currently shown rule-set. Similarly, we also expect to program the modules interactively. This is made possible because the statements of the script can be distributed to the modules and executed step-by-step. Once a satisfying script for the experiment at hand has been developed, it can be distributed to the modules and executed on the virtual machine independently from the development workstation.

V. Vision

We now present an overall vision of how we imagine developing autonomous controllers using the Modular Commander framework. As an example, consider developing the autonomous controller for docking of ATRON vehicles (the example of Section IV-B). The programmer starts by assembling two ATRON vehicles and defining gait tables for behavior sequences such as moving, turning, docking, and undocking. Using the GUI interface the programmer interactively experiments with and refines the gait tables, until they provide the required behavior. The programmer then uses the GUI to inspect various sensor values, such as the proximity sensors when two cars are close to each other, to determine when certain behaviors should be triggered. Using this information, the programmer can incrementally start to define simple reflexes: a condition on a sensor value and the corresponding sequence of behaviors that should be triggered. At this point the programmer is working interactively with the robot, while trying to minimize the amount of interaction required to make the robots behave as required. As soon as a condition or a behavior is modified, it is deployed to the running robot, providing immediate feedback.

Reflexes are specified using a simple pairing of conditions and behaviors that are distributed throughout the robot. These condition-behavior pairs can be structured, using a distributed interpreter approach. In general, the reflexes can be augmented with higher-level behaviors such as random exploration using a layered approach [21], resulting in a working system for the specific module configuration. To make the resulting program independent of the hardware IDs (modules, sensors, actuators, etc.) and the specific rotation of modules, the programmer must abstract over hardware IDs using labels that are defined symbolically using information such as the number of connections and the relative orientation or distance to some other module. As the programmer is defining criteria for each label, the system can interactively indicate what modules satisfy the criteria, either in terms of physical modules or using e.g. the simulator. The labels effectively allow the programmer to abstract over the concrete shape of the robot, meaning that if a robot with the same appearance is later assembled it will have the same behavior.

The system is not intended to scale to large numbers of modules. Indeed, we assume that the programmer will need to work at different scales in scenarios with many modules, for example by programming meta-modules using the Modular Commander framework and then relying on the properties of meta-modules to scale to larger numbers of modules [22]. Nevertheless, we could imagine Modular Commander also supporting development at the level of meta-modules, or more generally at any level of a hierarchical robot [4], potentially providing scalability to significantly larger robots.

VI. Acknowledgements

This research is funded by the Danish Council for Technology and Production through the project “Morphing Production Lines”. The authors also want to thank the people

who have contributed with ideas and suggestions: David Brandt, Danish Shaikh, Lars Kristian Lindegaard Mikkelsen, and Jørgen C. Larsen.

REFERENCES

[1] U. Schultz, "Unified Simulator for Self-reconfigurable Robots (USSR)," http://modular.mmmi.sdu.dk/wiki/USSR.

[2] D. Christensen, U. Schultz, D. Brandt, and K. Stoy, "A unified simulator for self-reconfigurable robots," in *Proc., The IEEE/RSJ International Conference on Intelligent Robots and Systems*, Nice, France, 2008, (to appear).

[3] M. W. Jorgensen, E. H. Ostergaard, and H. H. Lund, "Modular ATRON: Modules for a self-reconfigurable robot," in *Proceedings of IEEE/RSJ International Conference on Robots and Systems (IROS)*, Sendai, Japan, Sep. 2004, pp. 2068–2073.

[4] K. Stoy, A. Lyder, R. F. Mendoza Garcia, and D. J. Christensen, "Hierarchical Robots," in *Proc. of the IROS 2007 Workshop on Self-Reconfigurable Robots/Systems and Applications*, 2007.

[5] A. Lyder, R. F. Mendoza Garcia, and K. Stoy, "Mechanical Design of Odin, an Extendable Heterogeneous Deformable Modular Robot," in *Proc., The IEEE/RSJ International Conference on Intelligent Robots and Systems*, Nice, France, 2008, (to appear).

[6] M. Bordignon, U. P. Schultz, and L. Lindegaard Mikkelsen, "Implementing flexible parallelism for modular self-reconfigurable robots," 2008, submitted.

[7] D. A. Rusling. Kernel mechanisms: Bottom half handling. [Online]. Available: http://tldp.org/LDP/tlk/kernel/kernel.html

[8] TinyOS Community Forum. [Online]. Available: http://www.tinyos.net/

[9] V. Zykov, A. Chan, and H. Lipson, "Molecubes: an open-source modular robotics kit," in *Proc. IROS'07 Workshop on Self-Reconfigurable Robots & Systems and Applications*, San Diego, CA, USA, 2007.

[10] D. Brandt, J. Larsen, D. Christensen, R. Garcia, D. Shaikh, U. Schultz, and K. Stoy, "Flexible, fpga-based electronics for modular robots," 2008, submitted.

[11] The Contiki Operating System. [Online]. Available: http://www.sics.se/contiki/

[12] D. J. Christensen, M. Bordignon, U. P. Schultz, D. Shaikh, and K. Stoy, "Morphology independent learning in modular robots," in *Proc., International Symposium on Distributed Autonomous Robotic Systems (DARS)*, 2008, (submitted for review).

[13] R. Garcia, K. Stoy, D. Christensen, and A. Lyder, "A self-reconfigurable communication network for modular robots," in *Proceedings of the First International Conference on Robot Communication and Coordination (ROBOCOMM2007)*. ACM, Oct. 2007.

[14] S. Murata, E. Yoshida, K. Tomita, H. Kurokawa, A. Kamimura, and S. Kokaji, "Hardware design of modular robotic system," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Takamatsu, Japan, 2000, pp. 2210–2217.

[15] M. Yim, "Locomotion with a unit-modular reconfigurable robot," Ph.D. dissertation, Department of Mechanical Engineering, Stanford University, 1994.

[16] U. Schultz, "Distributed control diffusion: Towards a flexible programming paradigm for modular robots," in *Proceedings of the First International Conference on Robot Communication and Coordination (ROBOCOMM2007)*. ACM, Oct. 2007.

[17] D. Brandt and E. Ostergaard, "Behaviour subdivision and generalization of rules in rule based control of the ATRON self-reconfigurable robot," in *Proceeding of the International Symposium on Robotics and Automation (ISRA)*, Queretaro, Mexico, Sep. 2004, pp. 67–74.

[18] U. Schultz, D. Christensen, and K. Stoy, "A domain-specific language for programming self-reconfigurable robots," in *APGES 2007 — Automatic Program Generation for Embedded Systems — Workshop Proceedings*, Oct. 2007, pp. 28–36.

[19] K. Stoy, W.-M. Shen, and P. Will, "Implementing configuration dependent gaits in a self-reconfigurable robot," in *Proceedings of the 2003 IEEE international conference on robotics and automation (ICRA'03)*, Tai-Pei, Taiwan, Sep. 2003, pp. 3828–3833.

[20] K. Stoy, D. Christensen, D. Brandt, and U. Schultz, "Exploit morphology to simplify docking of self-reconfigurable robots," in *Proc., The 9th Int. Symposium on Distributed Autonomous Robotic Systems*, Tsukuba, Japan, 2008, (submitted for review).

[21] R. Brooks, "A robust layered control system for a mobile robot," *IEEE Journal of Robotics and Automation*, vol. 2, pp. 14–23, Mar. 1986.

[22] D. Christensen and K. Stoy, "Selecting a meta-module to shape-change the ATRON self-reconfigurable robot," in *Proceedings of IEEE International Conference on Robotics and Automations (ICRA)*, Orlando, USA, May 2006, pp. 2532–2538.